BSTZ No. 42P14357
Express Mail No. EV323392793

UNITED STATES PATENT APPLICATION

FOR

DATA PACKET ARITHMETIC LOGIC DEVICES AND METHODS

Inventors:

Corey Gee
Bapiraju Vinnakota
Saleem Mohammadali
Carl A. Alberola

Prepared by:

Blakely, Sokoloff, Taylor & Zafman LLP
12400 Wilshire Boulevard, Suite 700
Los Angeles, California 90025
(714) 557-3800

DATA PACKET ARITHMETIC LOGIC DEVICES AND METHODS

FIELD

[0001] This disclosure relates generally to data packet manipulation, and specifically, to new instruction definitions for a packet add (PADD) operation and for a single instruction multiple add (SMAD) operation, to a new PADD logic device that performs the PADD operation, and to a new SMAD logic device that performs the SMAD operation.

BACKGROUND

[0002] Many applications require the manipulation of data residing in data packets. For instance, packet processing in voice applications require the manipulation of several layers of protocol headers and several types of protocols. Also, protocols such as Internet Protocol (IP), Asynchronous Transfer Mode (ATM), and ATM adaptation layers (AALs) require header manipulation and error detection.

[0003] In the prior art, reduced instruction set computation (RISC) processors are used to perform manipulation of packet data. However, processors typically require many clock cycles to perform complex data manipulation. In addition, because processors typically operate on fixed length words, some inefficiencies result when the data to be manipulated is less than or more than the length of the word.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] Figure 1 illustrates a block diagram of an exemplary packet arithmetic logic device in accordance with an embodiment of the invention;

[0005] Figure 2A illustrates an exemplary syntax for an instruction to perform a packet addition (PADD) in accordance with another embodiment of the invention;

[0006] Figure 2B illustrates various examples of PADD instructions in accordance with another embodiment of the invention;

[0007] Figure 3A illustrates an exemplary syntax for an instruction to perform a single multiple data add (SMAD) in accordance with another embodiment of the invention;

[0008] Figure 3B illustrates various examples of SMAD instructions in accordance with another embodiment of the invention;

[0009] Figure 4 illustrates diagrams of an exemplary pair of operand packets and a result packet undergoing a packet addition (PADD) function in accordance with another embodiment of the invention;

[0010] Figure 5 illustrates a block diagram of an exemplary PADD logic device that performs the PADD function in accordance with another embodiment of the invention;

[0011] Figure 6 illustrates a table listing of exemplary 32-bit length masks used in the exemplary PADD logic device that performs the PADD function in accordance with another embodiment of the invention;

[0012] Figure 7 illustrates a block diagram of an exemplary single multiple data add (SMAD) logic device in accordance with another embodiment of the invention;

[0013] Figure 8 illustrates an exemplary block diagram of a 32-bit carry-save adder (CSA) in accordance with an embodiment of the invention;

[0014] Figure 9 illustrates an exemplary block diagram of a 16-bit CSA in accordance with an embodiment of the invention;

[0015] Figure 10 illustrates an exemplary block diagram of a 8-bit CSA in accordance with an embodiment of the invention;

[0016] Figure 11 illustrates an exemplary table illustrating an aspect of the operation of the 32-bit CSA in accordance with an embodiment of the invention;

[0017] Figure 12 illustrates an exemplary table illustrating an aspect of the operation of the 16-bit CSA in accordance with an embodiment of the invention; and

[0018] Figure 13 illustrates an exemplary table illustrating an aspect of the operation of the 8-bit CSA in accordance with an embodiment of the invention.

DETAILED DESCRIPTION

I. Packet Arithmetic Logic Device

[0019] Figure 1 illustrates a block diagram of an exemplary packet arithmetic logic device 100 in accordance with an embodiment of the invention. The packet arithmetic logic device 100 performs various operations on data packets. Such operations include packet processing for voice applications which require the manipulation of several layers of protocol headers and several types of protocols, and header manipulation and error detection especially in complex protocols such as Internet protocol (IP), asynchronous transfer mode (ATM), and

ATM adaptation layers (AALs). The packet arithmetic logic device 100 performs these operations in substantially less clock cycles than the prior art processors which take a multitude of steps to achieve these operations.

[0020] The packet arithmetic logic device 100 comprises an instruction control device 102, a result register RZ 104, a plurality of source data registers RX 106, RY, 108, RX+1 110, and RY+1 112, a data bus 114, a packet add (PADD) logic device 116, and a single instruction multiple data add (SMAD) logic device 118. The instruction control device 102 receives, interprets, and controls the registers and logic devices to properly execute the instruction. The result data register RZ 104 stores the result of the packet arithmetic operations. The source data registers RX 106, RY, 108, RX+1 110, and RY+1 112 store the various operands for the packet arithmetic operations. The PADD logic device 116 performs a packet add operation in about one to two clock cycles. The SMAD logic device 118 performs a multiple data add operation in about one to two clock cycles.

[0021] The following provides various instruction definitions which the packet arithmetic logic device 100 interprets in performing the specified packet arithmetic operations.

II. Arithmetic Instructions for Packet Processing

II-A. PADD Instruction

[0022] Figure 2A illustrates an exemplary syntax for an instruction 200 to perform a packet addition (PADD) in accordance with another embodiment of the invention. In a PADD function, at least a portion of an operand packet X stored in register RX is added with at least another portion of an operand packet Y stored in register RY or an immediate operand to form a result packet Z stored in result register RZ. Optionally, a carry in bit, set by a previous instruction, may be used as a third operand in the addition.

[0023] The PADD instruction 200 defines the result register RZ and one or two source registers RX and RY. The PADD instruction 200 may also define an immediate value as an operand and designated in the instruction as <UI8: immediate>. The PADD instruction 200 may further define the start bit and stop bit of the data field to be modified. These are respectively designated as <UI5: start> and <UI5: stop>. The PADD instruction 200 may also include several control parameters, including a control parameter designated as [-C] to indicate an addition with a carry in, a control parameter designated as [-M] to indicate a

modulo $2^n - 1$ addition, a control parameter designated as -N to indicate an addition affecting only the specified data field, and a control parameter designated as -I to indicate that the second operand is supplied as an immediate value.

[0024] Figure 2B illustrates various examples of PADD instructions 250 in accordance with another embodiment of the invention. In the first example instruction:

PADD RZ, RX, RY

the instruction control device 102 causes the PADD logic device 116 to add the operand X stored in the source register RX 106 to the operand Y stored in the source register RY 108, and place the result Z in the result register RZ 104 (i.e. RZ = RX + RY).

In the second example instruction:

PADD -C RZ, RX, RY

the instruction control device 102 causes the PADD logic device 116 to add the operand X stored in the source register RX 106, the operand Y stored in the source register RY 108, and the carry in from a previous instruction, and place the result Z in the result register RZ 104 (i.e. RZ = RX + RY + Cin).

In the third example instruction:

PADD -I RZ, RX, <UI8: immediate>

the instruction control device 102 causes the PADD logic device 116 to add the operand X stored in the source register RX 106 to an immediate value specified in <UI8: immediate>, and place the result Z in the result register RZ 104 (i.e. RZ = RX + <immediate>).

In the fourth example instruction:

PADD -N RZ, RX, RY <UI5: start>, <UI5: stop>

the instruction control device 102 causes the PADD logic device 116 to add the data field beginning at the start bit and ending at the stop bit of operand X stored in the source register RX 106 to the data field beginning at the least significant bit and having a length defined as stop - start +1 in the operand Y stored in the source register RY 108, and place the result data field in the same bit position defined by the start and stop in the result Z stored in the result register RZ. The remaining data fields of operand X stored in source register RX 106 outside of the specified data field are copied bitwise to the result Z stored in result register RZ. (i.e. RZ = {RX[31:stop], (RX[stop:start] + RY[length]) modulo 2^{length}, RX[start:0]}).

In the fifth example instruction:

PADD -M RZ, RX, RY

the instruction control device 102 causes the PADD logic device 116 to modulo 2^n -1 add the operand X stored in the source register RX 106 to the operand Y stored in the source register RY 108, and place the result Z in the result register RZ 104 (i.e. RZ = (RX + RY) modulo 2^n -1).

In the sixth example instruction:

PADD -N -I RZ, RX, <UI8:immediate>, <UI5: start>

the instruction control device 102 causes the PADD logic device 116 to add the data field beginning at the start bit and ending at the most significant bit of operand X stored in the source register RX 106 to the data field beginning at the least significant bit and ending at the bit 31 - start bit of the immediate value, and place the result data field in the same bit position defined by the start and most significant bit of the result Z stored in the result register RZ 104. The remaining data field of operand X stored in source register RX 106 outside of the specified field is copied bitwise to the result Z stored in result register RZ 104. (i.e. RZ = {(RX[31:start] + immediate[31-start:0]) modulo $2^{31-start+1}$, RX[start:0]}).

II-B. SMAD Instruction

[0025] Figure 3A illustrates an exemplary syntax for an instruction 300 to perform a single instruction multiple data add (SMAD) in accordance with another embodiment of the invention. In a SMAD function, multiple operands are added together. For instance, if 32-bit addition is to be performed, up to four 32-bit operands X, Y, X+1, and Y+1 stored respectively in registers RX 106, RY 108, RX+1 110, and RY+1 112 may be added to form result Z stored in result register RZ 104. If 16-bit addition is to be performed, up to eight 16-bit operands X[15:0], X[31:16], Y[15:0], Y[31:16], X+1[15:0], X+1[31:16], Y+1[15:0], and Y+1[31:16] stored respectively as pairs in registers RX 106, RY 108, RX+1 110 and RY+1 112 may be added to form result Z stored in result register RZ 104. If 8-bit addition is to be performed, up to 16 8-bit operands X[7:0], X[15:8], X[23:16], X[31:24], Y[7:0], Y[15:8], Y[23:16], Y[31:24], X+1[7:0], X+1[15:8], X+1[23:16], X+1[31:24], Y+1[7:0], Y+1[15:8], Y+1[23:16], Y+1[31:24] stored respectively as quads in registers RX 106, RY 108, RX+1 110 and RY+1 112 may be added to form result Z stored in result register RZ 104.

[0026] The SMAD instruction 300 defines the result register RZ and one or two source registers RX and RY. The SMAD instruction 300 may also include several control parameters, including a control parameter designated as [-A] to indicate that the result is accumulated into the result register RZ, and a control parameter designated as [-M] to indicate a modulo 2^n - 1 addition. The SMAD instruction 300 may also include a parameter designated as <UI2: Length> that indicates the data width of the operands (e.g. 0 indicates 8-bit operand, 1 indicates 16-bit operands, and 2 indicates 32-bit operands). In addition, the SMAD instruction 300 may include a parameter designated as <U12: Num Ops> to indicate the number of operands to be used in the addition (e.g. 0 indicates two source operands RX and RY, 1 indicates three source operands RX, RX+1, and RY, 2 indicates three source operands RX, RY, and RY+1, and 4 indicates four operands RX, RY, RX+1, and RY+1).

[0027] Figure 3B illustrates various examples of SMAD instructions 350 in accordance with another embodiment of the invention. In the first example instruction:

SMAD RZ, RX, RY, 2, 0

the instruction control device 102 causes the SMAD logic device 118 to add the 32-bit operand X stored in the source register RX 106 to the 32-bit operand Y stored in the source register RY 108, and place the result Z in the result register RZ 104 (i.e. RZ = RX + RY).

In the second example instruction:

SMAD -A RZ, RX, RY

the instruction control device 102 causes the SMAD logic device 118 to add the 32-bit operand X stored in the source register RX 106, the 32-bit operand Y stored in the source register RY 108, and the 32-bit operand Z stored in the result register RZ 104, and place the result Z in the result register RZ 104 (i.e. RZ = RZ + RX + RY).

In the third example instruction:

SMAD RZ, RX, RY, 2, 3

the instruction control device 102 causes the SMAD logic device 118 to add the 32-bit operand X stored in the source register RX 106, the 32-bit operand Y stored in the source register RY 108, the 32-bit operand X+1 stored in the source register RX+1 110, and the 32-bit operand Y+1 stored in the source register RY+1 112, and place the result Z in the result register RZ 104 (i.e. RZ = RX + RY + RX+1 +RY+1).

In the fourth example instruction:

SMAD RZ, RX, RY, 0, 0

the instruction control device 102 causes the SMAD logic device 118 to add the 8-bit operand X[7:0] stored in the source register RX[7:0] 106, the 8-bit operand X[15:8] stored in the source register RX[15:8] 106, the 8-bit operand X[23:16] stored in the source register RX[23:16] 106, the 8-bit operand X[31:24] stored in the source register RX[31:24] 106, the 8-bit operand Y[7:0] stored in the source register RY[7:0] 108, the 8-bit operand Y[15:8]

stored in the source register RY[15:8] 108, the 8-bit operand Y[23:16] stored in the source register RY[23:16] 108, and the 8-bit operand Y[31:24] stored in the source register RY[31:24] 108, and place the result Z in the result register RZ 104 (i.e. RZ = RX[7:0] + RX[15:8] + RX[23:16] + RX[31:24] + RY[7:0] + RY[15:8] + RY[23:16] + RY[31:24]).

In the fifth example instruction:

SMAD -M RZ, RX, RY, 2, 0

the instruction control device 102 causes the SMAD logic device 118 to modulo 2^n -1 add the 32-bit operand X stored in the source register RX 106 to the 32-bit operand Y stored in the source register RY 108, and place the result Z in the result register RZ 104 (i.e. RZ = (RX + RY) modulo 2^n -1).

In the sixth example instruction:

PADD -A -M RZ, RX, RY, 2, 0

the instruction control device 102 causes the SMAD logic device 118 to modulo 2^n -1 add the 32-bit operand X stored in the source register RX 106, the 32-bit operand Y stored in the source register RY 108, and the 32-bit operand Z stored in the result register RZ 104, and place the result Z in the result register RZ 104 (i.e. RZ = (RZ + RX + RY) modulo 2^n -1).

I. <u>The PADD Logic Device</u>

[0028] Figure 4 illustrates diagrams of exemplary pair of operand packets and a result packet undergoing a packet addition (PADD) function in accordance with another embodiment of the invention. In a PADD function, an operand data field in an operand packet X is to be added with another operand data field in operand packet Y to form a data field in a result packet Z. The operand data field in the operand packet X has a length of n bits and its least significant bit is situated m bits from the least significant bit of the operand packet X. The operand data field in the operand packet Y also has a length of n bits and its least significant

bit coincides with the least significant bit of the operand packet Y. The result data field in the result packet Z has a length of n bits and its least significant bit is situated m bits from the least significant bit of the operand packet Z. The remaining data fields in the operand packet X are copied bitwise to the result packet Z.

[0029] For example, operand data Field X-2 in operand packet X, which has a length of n bits and its least significant bit is situated m bits from the least significant bit of the operand packet X, is added to operand data Field Y-1 which also has a length of n bits and its least significant bit coincides with the least significant bit of the operand packet Y. The result data field Z-2, being the addition of Fields X-2 and Field Y-1, has a length of n bits and its least significant bit is situated m bits from the least significant bit of the operand packet Z. The remaining data Fields X-1 and X-3 in the operand packet X are copied bitwise to data Fields Z-1 and Z-3 of the result packet Z.

[0030] Figure 5 illustrates a block diagram of an exemplary PADD logic device 500 that performs the PADD function in accordance with another embodiment of the invention. The PADD logic device 500 comprises a left shifter 502, bitwise logic ORs 504 and 514, bitwise logic ANDs 506, 510 and 512, and an adder 508. The operand Y is applied to the input of the left shifter 502 and the number m controls the amount of left shifting of the left shifter 502. The output of the left shifter 502 is applied to an input of the bitwise OR 504.

[0031] A mask (m + n) as listed in the table shown in Figure 6 is applied to the other input of the bitwise OR 504, to an inverted input of bitwise AND 506, to an inverted input of bitwise AND 510, and to an input of bitwise AND 512. If the carry-in C_{in} is 1, the left shifter 502 shifts in logic ones at its least significant bits, otherwise it shifts logic zeros. The operand X is applied to the respective inputs of the bitwise ANDs 506 and 512. The outputs of the bitwise OR 504 and bitwise AND 506 are applied to the inputs of adder 508. The output of the adder 508 is applied to the input of bitwise AND 510. The outputs of bitwise ANDs 510 and 512 are applied to the inputs of bitwise OR 514. And, the output of bitwise OR 514 generates the result packet Z.

[0032] The bitwise OR logic device 504 generates an intermediate packet 550 comprising the operand data Field Y-1 situated at the same bit position as the operand data Field X-1, with logic ones on the more significant bit side of the Field Y-1, and with either all logic ones if the carry-in C_{in} is asserted or all logic zeros if the carry-in C_{in} is not asserted on the lesser

significant bit side of the Field Y-1. Thus, the following relationship holds for the output of the bitwise OR logic device 504:

Field X-3 FieldX-2 Field X-1 Operand X
1...........1 FieldY-1 0...........0 Intermediate Packet 550 $C_{in} = 0$
1...........1 FieldY-1 1...........1 Intermediate Packet 550 $C_{in} = 1$

[0033] The intermediate packet 550 having logic ones at the same bit position as Field X-1 allows the carry-in to propagate to the sum field X+Y.

[0034] The bitwise AND logic device 506 generates an intermediate packet 552 which comprises logic zeros at the bit position of Field X-3 and Fields X-2 and X-1 at the same bit position as the corresponding Fields X-2 and X-1 of the operand packet X. Thus, the following relationship holds for the output of the bitwise AND logic device 506:

Field X-3 Field X-2 Field X-1 Operand X
0...........0 Field X-2 Field X-1Intermediate Packet 552

[0035] The output of the adder 508 generates an intermediate packet 554 which comprises don't cares x at the same bit position as Field X-3, the sum Field X+Y at the same bit position as Field X-2, and the Field X-1 at the same bit position as Field X-1. Thus, the following relationship holds for the output of the adder 508:

Field X-3 Field X-2 Field X-1 Operand X
x...........x Field X+Y Field X-1 Intermediate Packet 554

[0036] The bitwise AND logic device 510 generates an intermediate packet 556 which comprises logic zeros at the same bit position as Field X-3, the sum Field X+Y at the same bit position as Field X-2, and the Field X-1 at the same bit position as Field X-1. Thus, the following relationship holds for the output of the bitwise AND logic device 510:

Field X-3 Field X-2 Field X-1 Operand X
0...........0 Field X+Y Field X-1 Intermediate Packet 556

[0037] The bitwise AND logic device 512 generates an intermediate packet 558 which comprises Field X-3 at the same bit position as Field X-3 and logic zeros at the same bit position as Fields X-1 and X-2. Thus, the following relationship holds for the output of the bitwise AND logic device 512:

Field X-3 Field X-2 Field X-1	Operand X
Field X-3 Field 0....................0	Intermediate Packet 558

[0038] The bitwise OR logic device 514 bitwise ORs the outputs of the bitwise AND logic device 510 and 512 to generate the result packet Z.

[0039] The following operand packets and result packet serves as an example to illustrate the operation of the PADD logic device 500:

0...111101101001101011010 Operand X
0...000000000000000011001 Operand Y
0...111101110110001011010 Result Z
$m = 8$, $n = 8$, $C_{in} = 0$

[0040] As discussed above, the operand Y is applied to the input of the left shifter 502, the number m controls the amount of left shifting, and the carry-in C_{in} causes the left shifter 502 to shift in logic ones if it is asserted and logic zeros if it is not asserted. In this example, the number m is eight (8) and the C_{in} is a logic zero (0). Therefore, the left shifter 502 left shifts the operand Y by eight (8) bits and shifts in logic zeros (0s). Accordingly, the output of the left shifter 502 is as follows:

0...00000001100100000000 (i.e. 0...000 Field Y-1 00000000)

[0041] Referring to both Figures 5 and 6, in this example the number (m + n) is equal to 16. Therefore, according to the table, the mask and its complement are given by the following:

mask = 1...11100000000000000000

complement mask = 0...0001111111111111111

[0042] The output of the bitwise OR 504, being the bitwise OR of the output of the left shifter 502 and the mask, is given by the following:

1...1110000001100100000000 (i.e. 1...111 Field Y-1 00000000)

[0043] The output of the bitwise AND 506, being the bitwise AND of the complement mask and the operand X, is given by the following:

0...0000011010011010110l0 (i.e. 0...000 Field X-2 Field X-1)

[0044] The outputs of the bitwise OR 504 and the bitwise AND 506 are summed by the adder 508. Since the carry-in C_{in} is a logic zero, the output of the adder 508 is given by the following:

1...1111111001000101l010 (i.e. 1...111 Field X+Y Field X-1)

[0045] The output of the adder 508 and the complement mask are bitwise AND by bitwise AND logic device 510. Therefore, the output of the bitwise AND logic device 510 is given by the following:

0...0000011100100010l1010 (i.e. 0...000 Field X+Y Field X-1)

[0046] The output of the bitwise AND 512, being the bitwise AND of the complement mask and the operand X, is given by the following:

0...1111000000000000000000 (i.e. Field X-3 0000000000000000)

[0047] The output of the bitwise OR logic device 514, which is the bitwise OR of the output of the bitwise AND logic devices 510 and 512, is given by the following:

0...1111011101100010 11010 (i.e. Field X-3 Field X+Y Field X-1)

which is the result packet Z.

[0048] An advantage of the PADD logic device 500 is that it performs the PADD operation relatively fast and efficient. In the prior art, RISC processors are employed to perform the PADD operation. However, RISC processors need to perform many logic operations to perform the PADD operation. This requires the RISC processors to take numerous clock cycles to perform the operation. With the PADD logic device 500, only one or two processor cycles are used to attain the PADD result.

II. Modulo 2^n / Modulo 2^n - 1 Addition

[0049] Figure 7 illustrates a block diagram of an exemplary single multiple data add (SMAD) logic device 700 in accordance with another embodiment of the invention. The SMAD logic device performs the modulo 2^n and/or the modulo 2^n - 1 of up to four 32-bit numbers, eight 16-bit numbers, or 16 8-bit numbers. The SMAD logic device 700 comprises a 32-bit carry-save adder (CSA) 702, a 16-bit CSA 704, and a 8-bit CSA 706. The SMAD logic device 700 further comprises a 6-input/2-output multiplexer 708, a first 32-bit adder 710, a second 32-bit adder 712, 2-input/1-output multiplexers 714, 716, and 718, and 3-input/1-output multiplexer 720.

[0050] The 32-bit CSA 702 receives up to four 32-bit operands X_0[31:0], X_1[31:0], Y_0[31:0], and Y_1[31:0], and generates a carry C<32:0> and a save S<31:0>. The 32-bit CSA 702 comprises 32 4:2 compressors 702-0 through 702-31. Each of the 4:2 compressors, represented as 702-n, receives as inputs X_0[n], X_1[n], Y_0[n], and Y_1[n], and generates the carry C<n> and save S<n>. The carry of compressor 702-n is allowed to carry to the first compressor 702-0 except under certain circumstances with regard to modulo 2^n addition, as will be explained further below.

[0051] The 16-bit CSA 704 receives four operands C<31:16>, C<15:1>, S<31:16>, and S<15:0> from the carry C<31:0> and the save S<31:0> of the 32-bit CSA 702 if 16-bit addition is being performed, and generates carry C1<15:1> and save S1<15:0>. The 16-bit CSA 704 comprises 16 4:2 compressors 704-0 through 704-15. Each of the 4:2 compressors, represented as 704-n, receives as inputs C<n>, S<n> except that of C<0> which instead

receives a logic zero, and generates the carry C1<n> and save S1<n>. The carry of compressor 704-n is allowed to carry to the first compressor 704-0 except under certain circumstances with regard to modulo 2^n addition, as will be explained further below.

[0052] The 8-bit CSA 706 receives four operands C1<15:8>, C1<7:1>, S1<15:8>, and S1<7:0> from the carry C1<15:1> and the save S1<15:0> of the 16-bit CSA 704 if 8-bit addition is being performed, and generates carry C2<7:1> and save S2<7:0>. The 8-bit CSA 706 comprises eight 4:2 compressors 706-0 through 706-7. Each of the 4:2 compressors, represented as 706-n, receives as inputs C1<n>, S1<n> except that of C1<0> which instead receives a logic zero, and generates the carry C2<n> and save S2<n>. The carry of compressor 706-n is allowed to carry to the first compressor 706-0 except under certain circumstances with regard to modulo 2^n addition, as will be explained further below.

[0053] The six inputs to the 6-input/2-output multiplexer 708 include {24'hffffff, C2<7:1>, C<32>}, {16'hffff, C1<15:1>, C<32>}, {C<31:1>, C<32>}, {24'h0, S2<7:0>}, {16'h0, S1<15:0>}, and S<31:0>. If 32-bit addition is being performed, the multiplexer 708 selects as its outputs C<31:0> and S<31:0>. If 16-bit addition is being performed, the multiplexer 708 selects as its outputs {16'hffff, C1<15:1>, C<32>} and {16'h0, S1<15:0>}. If 8-bit addition is being performed, the multiplexer 708 selects as its outputs {24'hffffff, C2<7:1>, C<32>} and {24'h0, S2<7:0>}.

[0054] The outputs of the multiplexer 708 are applied in parallel to the respective inputs of the first and second 32-bit adders 710 and 712. The first 32-bit adder 710 has a logic zero as a carry-in C_{in}. The carry-out C_{out} of the first 32-bit adder 710 controls the multiplexers 714, 716, and 718 in a manner that if the carry-out C_{out} is asserted, the multiplexers 714, 716, and 718 select the corresponding sum_1 input, otherwise it selects the corresponding sum_0 input. The first 32-bit adder generates the sum_0 output, which is applied to the corresponding inputs of multiplexers 714, 716, and 718 if 8-bit, 16-bit, or 32-bit addition respectively is performed.

[0055] The second 32-bit adder 712 has a logic one as a carry-in C_{in}, and generates the sum_1 output, which is applied to the corresponding inputs of multiplexers 714, 716, and 718 if 8-bit, 16-bit, or 32-bit addition respectively is performed. The outputs of the multiplexers 714, 716, and 718 are applied to the inputs of the 3-input/1-output multiplexer 720. If 8-bit addition is being performed, the multiplexer 720 selects as its output the output of multiplexer 714. If 16-bit addition is being performed, the multiplexer 720 selects as its

output the output of multiplexer 716. If 32-bit addition is being performed, the multiplexer 720 selects the output of multiplexer 718. The output of the multiplexer 720 is the result Z<31:0> of the modulo 2^n or modulo 2^n - 1 addition of the input operands. The following explains, in more detail, the various additions and operands that the modulo logic device 700 performs.

[0056] Figure 8 illustrates an exemplary block diagram of a 32-bit carry-save adder (CSA) 702 in accordance with an embodiment of the invention. As previously discussed with reference to Figure 7, the 32-bit CSA 700 comprises 32 4:2 compressors 702-0 sequentially through 702-31. The inputs to the 4:2 compressor 702-n includes operands y[n], y1[n], x[n], and x1[n]. The 4:2 compressor 702-n generates carry c[n] and save s[n]. The carry-out co_<n+1> of 4:2 compressor 702-n is coupled to the carry-in of 4:2 compressor 702-<n+1>, except that of compressor 702-31 whose carry-out is coupled to the carry-in of 4:2 compressor 702-0.

[0057] Figure 9 illustrates an exemplary block diagram of a 16-bit carry-save adder (CSA) 704 in accordance with an embodiment of the invention. As previously discussed with reference to Figure 7, the 16-bit CSA 704 comprises 16 4:2 compressors 704-0 sequentially through 704-15. The inputs to the 4:2 compressor 704-n include the output carry and save from the 32-bit CSA 702, such as s<n+16>, c<n+16>, s<n>, and c<n> except the first compressor 704-0 which has as inputs c<16>, s<16>, s<0>, and co1<16> from the carry out of the last compressor 704-15. The 4:2 compressor 704-n generates carry c1<n> and save s1<n>. The carry-out co1_<n+1> of 4:2 compressor 704-n is coupled to the carry-in of 4:2 compressor 704-<n+1>, except that of compressor 704-15 whose carry-out co1_16 is coupled to the carry-in of 4:2 compressor 704-0.

[0058] Figure 10 illustrates an exemplary block diagram of a 8-bit carry-save adder (CSA) 706 in accordance with an embodiment of the invention. As previously discussed with reference to Figure 7, the 8-bit CSA 706 comprises eight (8) 4:2 compressors 706-0 sequentially through 706-7. The inputs to the 4:2 compressor 706-n include the output save and carry from the 16-bit CSA 702, namely s1<n+8>, c1<n+8>, s1<n>, and c1<n> except the first compressor 706-0 which has as inputs s1<8>, c1<8>, s1<0>, and c2<8> from the carry of the last compressor 706-7. The 4:2 compressor 706-n generates carry c1<n> and save s1<n>. The carry-out co2_<n+1> of 4:2 compressor 706-n is coupled to the carry-in of 4:2

compressor 706-<n+1>, except that of compressor 706-7 whose carry-out co2_8 is coupled to the carry-in of 4:2 compressor 706-0.

II-A 32-bit Operands Modulo 2^n Addition

[0059] With reference to the table illustrated in Figure 11, if the 32-bit operands $X_0[31:0]$, $X_1[31:0]$, $Y_0[31:0]$, and $Y_1[31:0]$ are applied to the 32-bit CSA 702, the carry-out co_32 of the last 4:2 compressor 702-31 does not propagate to the carry-in of the first 4:2 compressor 702-0. In addition, the carry C<32> of the last 4:2 compressor does not propagate to the multiplexer 708. Since this is a 32-bit operation, the multiplexer 708 selects as its outputs the carry C<31:1> and save S<31:0>. Accordingly, the carry C<31:1> and save S<31:0> are summed by the first adder 710 to generate sum_0. The second adder 712 is ignored in modulo 2^n addition. The multiplexer 718 selects as its output the sum_0 for modulo 2^n addition. Since, again this is a 32-bit operation, the multiplexer 720 selects the output of the multiplexer 718. The output of the multiplexer 720 is the modulo 2^n addition of the operands $X_0[31:0]$, $X_1[31:0]$, $Y_0[31:0]$, and $Y_1[31:0]$.

II-B 32-bit Operands Modulo 2^n - 1 Addition

[0060] In the case of modulo 2^n - 1 addition of 32-bit operands $X_o[31:0]$, $X_1[31:0]$, $Y_o[31:0]$, and $Y_1[31:0]$, the carry-out of the last compressor 4:2 702-31 of the 32-bit CSA propagates to the carry-in of the first 4:2 compressor 702-0. In addition, the carry C[32] of the last 4:2 compressor 702-31 propagates to the multiplexer 708. Since this is a 32-bit operation, the multiplexer 708 selects as its outputs the {C<31:1>, C<32>} and save S<31:0>. Accordingly, the {C<31:1>, C<32>} and S<31:0> are summed by both the first and second adders 710 and 712 to generate respectively sum_0 and sum_1. If the carry out C_{out} of the first adder 710 is a logic one, the multiplexer 718 selects as its output the sum_1, otherwise it selects the sum_0. Since, again this is a 32-bit operation, the multiplexer 720 selects the output of the multiplexer 718. The output of the multiplexer 720 is the modulo 2^n -1 addition of the operands $X_0[31:0]$, $X_1[31:0]$, $Y_0[31:0]$, and $Y_1[31:0]$.

II-C 16-bit Operands Modulo 2^n Addition

[0061] With reference to the table illustrated in Figure 11, if eight (8) 16-bit operands $X_0[15:0]$, $X_0[31:16]$, $X_1[15:0]$, $X_1[31:16]$, $Y_0[15:0]$, $Y_0[31:16]$, $Y_1[15:0]$, and $Y_1[31:16]$ are applied to the 32-bit CSA 702, the carry-outs co_16 and co _32 of the 16th and last 4:2 compressors 702-15 and 702-31 do not propagate respectively to the carry-ins of the 17th and first 4:2 compressors 702-16 and 702-0. In addition, the carrys C<16> and C<32> generated by the 16th and last compressors 702-15 and 702-31 do not propagate to an input of the first compressor 704-0 of the 16-bit CSA 704 and to the multiplexer 708, respectively.

[0062] The carries C<31:16> and C<15:1,0> and saves S<31:16> and S<15:0> generated by the 32 bit- CSA 702 are applied to the 16-bit CSA 704, which generates carry C1<15:1,0> and save S1<15:0>. As shown in the table illustrated in Figure 12, the carry-out co1_16 and carry C1<16> of the last 4:2 compressor 704-15 do not propagate to the first 4:2 compressor 704-0.

[0063] Since this is a 16-bit operation, the multiplexer 708 selects as its outputs the {16'hffff, C1<15:1>, C<32>} and {16'h0, S1<15:0>}. Accordingly, the {16'hffff, C1<15:1>, C<32>} and {16'h0, S1<15:0>} are summed by the first adder 710 to generate sum_0. The second adder 712 is ignored in modulo 2^n addition. The multiplexer 716 selects as its output the sum_0<15:0> for modulo 2^n addition. Since, again this is a 16-bit operation, the multiplexer 720 selects the output of the multiplexer 716. The output of the multiplexer 720 is the modulo 2^n addition of the operands $X_0[15:0]$, $X_0[31:16]$, $X_1[15:0]$, $X_1[31:16]$, $Y_0[15:0]$, $Y_0[31:16]$, $Y_1[15:0]$, and $Y_1[31:16]$.

II-D <u>16-bit Operands Modulo 2^n - 1 Addition</u>

[0064] In the case of Modulo 2^n-1 addition of eight (8) 16-bit operands $X_0[15:0]$, $X_0[31:16]$, $X_1[15:0]$, $X_1[31:16]$, $Y_0[15:0]$, $Y_0[31:16]$, $Y_1[15:0]$, and $Y_1[31:16]$, the carry-outs co_16 and co_32 of the 16th and last 4:2 compressors 702-15 and 702-31 propagate respectively to the carry-ins of the 17th and first 4:2 compressors 702-16 and 702-0. In addition, the carries c<16> and c<31> generated by the 16th and last compressors 702-15 and 702-31 propagate to an input of the first compressor 704-0 of the 16-bit CSA 704 and to the multiplexer 708, respectively.

[0065] The carries C<31:16> and C<15:1,0> and saves S<31:16> and S<15:0> generated by the 32-bit CSA 702 are applied to the 16-bit CSA 704, which generates carry C1<15:1,0> and save S1<15:0>. The carry-out co1_16 and carry c1<16> of the last 4:2 compressor 704-15 propagate to the carry-in and input of the first 4:2 compressor 704-0 in modulo 2^n - 1 addition.

[0066] Since this is a 16-bit operation, the multiplexer 708 selects as its outputs the {16'hffff, C1<15:1>, C<32>} and {16'h0, S1<15:0>}. Accordingly, the {16'hffff, C1<15:1>, C<32>} and {16'ho, S1<15:0>} are summed by the first and second adders 710 and 712 to generate respectively sum_0<15:0> and sum_1<15:0>. If the carry out C_{out} of the first adder 710 is a logic one, the multiplexer 716 selects as its output the sum_1<15:0>, otherwise it selects the sum_0<15:0>. Since, again this is a 16-bit operation, the multiplexer 720 selects the output of the multiplexer 716. The output of the multiplexer 720 is the modulo 2^n -1 addition of the operands $X_0[15:0]$, $X_0[31:16]$, $X_1[15:0]$, $X_1[31:16]$, $Y_0[15:0]$, $Y_0[31:16]$, $Y_1[15:0]$, and $Y_1[31:16]$.

II-E <u>8-bit Operands Modulo 2^n Addition</u>

[0067] With reference to the table illustrated in Figure 11, if the 16 8-bit operands $X_0[7:0]$, $X_0[15:8]$, $X_0[23:16]$, $X_0[31:24]$, $X_1[7:0]$, $X_1[15:8]$, $X_1[23:16]$, $X_1[31:24]$, $Y_0[7:0]$, $Y_0[15:8]$, $Y_0[23:16]$, $Y_0[31:24]$, $Y_1[7:0]$, $Y_1[15:8]$, $Y_1[23:16]$, and $Y_1[31:24]$ applied to the 32-bit CSA 702 are to be modulo 2^n added, the carry-outs c0_8, c0_16, co_24 and co_32 of 4:2 compressors 702-7, 702-15, 702-23, and 702-31 do not propagate respectively to the carry-ins of 4:2 compressors 702-8, 702-16, 702-24, and 702-0. In addition, the carries c<8>, c<16>, c<24>, and c<32> of respectively 4:2 compressors 702-7, 702-15, 702-23, and 702-31 do not propagate respectively to the inputs to the 4:2 compressors 704-8, 704-0 and 704-8, and multiplexer 708.

[0068] The carries C<7:1, 0>, C<15:8>, C<23:16> and C<31:24>, and saves S<7:0>, S<15:8>, S<23:16> and S<31:24> are applied to the 16-bit CSA 704, which generates carries C1<7:1,0> and C1<15:8> and saves S1<7:0> and S1<15:8>. With reference to the table illustrated in Figure 12, the carry-outs co1_8 and co1_16 of 4:2 compressors 704-7 and 704-15 do not propagate respectively to 4:2 compressors 704-8 and 704-0. In addition, the carrys

c1<8> and c1<16> of the 4:2 compressors 704-7 and 704-16 do not propagate respectively to 4:2 compressors 706-0 and 704-0.

[0069] The carries C1<7:1,0> and C1<15:8> and saves S1<7:0> and S1<15:8> are applied to the 8-bit CSA 706, which generates carry C2<7:1,0> and save S2<7:0>. With reference to the table illustrated in Figure 13, the carry-out co2_8 and carry c2<8> of the last 4:2 compressor 706-7 do not propagate to the carry-in and input of the first compressor 706-0.

[0070] Since this is an 8-bit operation, the multiplexer 708 selects as its outputs the {24'hffff, c2<7:1>, c<32>} and {24'h0, S2<7:0>}. Accordingly, the {24'hffff, c2<7:1>, c<32>} and {24'h0, S2<7:0>} are summed by the first adder 710 to generate sum_0<7:0>. The second adder 712 is ignored in modulo 2^n addition. The multiplexer 714 selects as its output the sum_0<7:0> for modulo 2^n addition. Since, again this is an 8-bit operation, the multiplexer 720 selects as its output the output of the multiplexer 714. The output of the multiplexer 720 is the modulo 2^n addition of the operands X_0[7:0], X_0[15:8], X_0[23:16], X_0[31:24], X_1[7:0], X_1[15:8], X_1[23:16], X_1[31:24], Y_0[7:0], Y_0[15:8], Y_0[23:16], Y_0[31:24], Y_1[7:0], Y_1[15:8], Y_1[23:16], and Y_1[31:24].

II-F <u>8-bit Operands Modulo 2^n - 1 Addition</u>

[0071] In the case of Modulo 2^n-1 addition of 16 8-bit operands X_0[7:0], X_0[15:8], X_0[23:16], X_0[31:24], X_1[7:0], X_1[15:8], X_1[23:16], X_1[31:24], Y_0[7:0], Y_0[15:8], Y_0[23:16], Y_0[31:24], Y_1[7:0], Y_1[15:8], Y_1[23:16], and Y_1[31:24], the carry-outs co_8, co_16, co_24, and co_32 of 4:2 compressors 702-7, 702-15, 702-23, and 702-31 do propagate respectively to the carry-ins of 4:2 compressors 702-8, 702-16, 702-24 and 702-0. Also, the carries c<8>, c<16>, c<24>, and c<32> do propagate respectively to the inputs of 4:2 compressors 704-8, 704-0, and 704-8, and to multiplexer 708.

[0072] The carries C<7:1, 0>, C<15:8>, C<23:16> and C<31:24>, and saves S<7:0>, S<15:8>, S<23:16> and S<31:24> are applied to the 16-bit CSA 704, which generates carries C1<7:1,0> and C1<15:8> and saves S1<7:0> and S1<15:8>. With reference to the table illustrated in Figure 12, the carry-outs co1_8 and co1_16 of 4:2 compressors 704-7 and 704-15 do propagate to 4:2 compressors 704-8 and 704-0. The carries C1<7:1,0> and C1<15:8> and saves S1<7:0> and S1<15:8> are applied to the 8-bit CSA 706, which generates carry C2<7:1,0> and save S2<7:0>. With reference to the table illustrated in Figure 13, the carry-

out co2_8 and carry c2<8> of the last 4:2 compressor 706-7 do propagate to the inputs of 4:2 compressor 706-0. The carry C2<7:1,0> and save S2<7:0> are applied to the multiplexer 708.

[0073] Since this is an 8-bit operation, the multiplexer 708 selects as its outputs the {24'hffffff, c2<7:1>, c<32>}, and {24'h0,S2<7:0>}. Accordingly, the {24'hffffff, c2<7:1>, c<32>}, and {24'h0,S2<7:0>} are summed by the first and second adders 710 and 712 to generate respectively sum_0<7:0> and sum_0<7:0>. If the carry out C_{out} of the first adder 710 is a logic one, the multiplexer 714 selects as its output the sum_1<7:0>, otherwise it selects the sum_0<7:0>. Since, again this is an 8-bit operation, the multiplexer 720 selects as its output the output of the multiplexer 714. The output of the multiplexer 720 is the modulo 2^n addition of the operands X_0[7:0], X_0[15:8], X_0[23:16], X_0[31:24], X_1[7:0], X_1[15:8], X_1[23:16], X_1[31:24], Y_0[7:0], Y_0[15:8], Y_0[23:16], Y_0[31:24], Y_1[7:0], Y_1[15:8], Y_1[23:16], and Y_1[31:24].

II-G <u>Conclusion - Modulo 2^n / Modulo 2^n - 1 Addition</u>

[0074] The modulo logic device 700 enables practical realization of implementing the SMAD/ESMAD functions. In the prior art, achieving the SMAD/ESMAD functions is typically done by executing a series of instruction by a processor. These instructions include a number of adds and logical operations, which can consume several to many clock processor cycles. The modulo logic device 700 can perform the SMAD/ESMAD functions within one or two processor cycles for substantial speedup in performance over executing instructions.

[0075] In the foregoing specification, this disclosure has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the embodiments of the embodiments of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.